Assessing Application Vulnerability to Radiation-Induced SEUs in Memory

Paul L. Springer
Jet Propulsion Laboratory
California Institute of Technology
Paul.Springer@jpl.nasa.gov

1. Introduction

One of the goals of the Remote Exploration and Experimentation (REE) project at JPL is to determine how vulnerable applications are to single event upsets (SEUs) when run in low radiation space environments using commercial-off-the-shelf (COTS) components [1]. Studies have shown that when bit flips are randomly injected into the processor or memory, simulating the effects of SEUs, a large percentage of them may have no effect on the program running [2].

Various explanations have been given to clarify the circumstances under which a fault does not affect program correctness. In Koga, et al [3], a description is given of the duty cycle of each register in the processor--the percentage of time during which the register contents are vulnerable to an SEU. Benso, et al [4], analyze why faults injected into either code or data space can be ineffective. He defines the *life period* for a program variable to be a period of time between when a value is written to the variable, and the last point at which that value is read by the processor. Any faults injected into the variable outside of a life period for the variable will be ineffective. Similarly, a fault injected into an instruction after the very last time that instruction is executed will have no effect.

This reasoning carries over into the analysis of the effects of SEUs on a program. Assuming a particular hardware configuration for a computer, and an environment where the level of radiation is fairly constant, the frequency with which SEUs affect the program RAM is proportional to the amount of RAM used and the length of time it is in use [5]. Therefore the important measure of a program's susceptibility to SEUs in this environment can be expressed in units of MB x seconds. This work defines such a measure and explores its usefulness.

2. Data Vulnerability

There can be several reasons why an SEU that occurs in the part of RAM dedicated to holding data has no effect on the processor or application. First, the fault might occur in a part of memory that is never accessed. Secondly, the location containing the flipped bit may have been previously used, but is no longer accessed after the fault occurs. A third possibility is that the faulty memory location is overwritten with correct data, after the fault occurs, but before it is read by the processor.

The idea of a variable's life period can be clarified by Figure 1. The variable pictured is first uninitialized, and not susceptible to an SEU. At the point in time labeled "w0", it is first written to, and remains vulnerable until the second read, r1. After r1, it is invulnerable until w2,

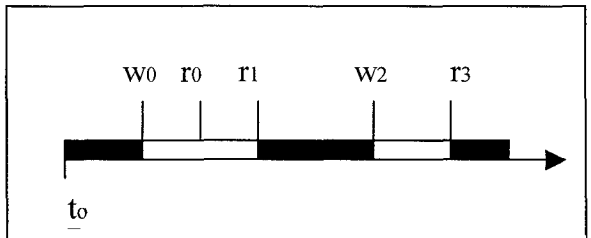

Figure 1. Vulnerability of a variable

and stays vulnerable until r3 occurs. The white portions in the figure are the periods in time of the variable's life period, when it is vulnerable to SEUs.

To formalize this, let us say that a byte of memory is vulnerable at any point in time if an SEU occurring at that time would be read by the processor. Another way of saying this is that a byte is vulnerable at any point during its life period. Define the *vulnerability* (expressed in units of byte-seconds) of that byte, $\mathbf{V}_b$, to be numerically equal to the number of seconds during an application run when it is vulnerable. Then the data vulnerability of an application, $\mathbf{V}_d$, is the sum of the $\mathbf{V}_b$ measure for each byte in the data segment of the application. It is most conveniently expressed in units of MB-seconds. Note that the vulnerability profile of an application can change depending on the input data and the actual path the program takes through its code. In this paper application

vulnerability will refer to a particular run of an application.

3. Code Vulnerability

The code vulnerability of an application, $\mathbf{V}_c$, can be calculated in a similar manner. Once an instruction is loaded into memory, it remains vulnerable until the last point in time at which it is read by the processor. Summing the vulnerabilities of each byte of code space will result in the $\mathbf{V}_c$ measure of the code for that application run.

The code vulnerability profile of an application plots the number of code bytes vulnerable at any instant of time against time. Figure 2 (not drawn to scale) shows the profile for a 1 MB application that runs through most its code sequentially and spends the remainder of its time (95 seconds) looping in the final 3 KB of code. The application runs immediately after being loaded, and is not run a second time unless it is reloaded. The profile function is non-increasing and its integral is $\mathbf{V}_c$, about 2.79 MB-seconds in this case.

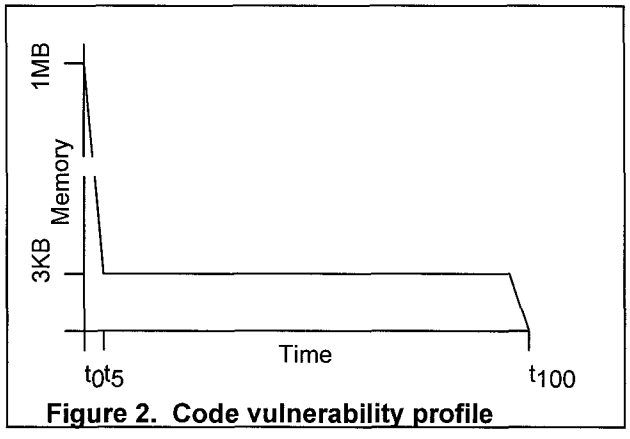

Figure 2. Code vulnerability profile

4. Vulnerability Tradeoffs

As long as code and data spaces are both in RAM, where there is an equal chance of SEU occurrence, the two vulnerabilities discussed can be combined to deduce total application vulnerability in data and code space. Application designers can use this information to predict how changes will affect overall application vulnerability.

Suppose, for example, a designer is considering the use of an Algorithm Based Fault Tolerant (ABFT) routine that would protect a 2 MB array from the affects of an SEU. This change would be made to the application which has the profile shown in Figure 2. Assume furthermore that the routine does not add a significant amount of code, but does cost 3 seconds of run time immediately after the program starts. Finally, let us say that the array to be protected is vulnerable for 1 second during the program run. The additional 3 seconds of run time of the ABFT routine will affect the left-most part of the profile shown, adding 3 MB-seconds to $\mathbf{V}_c$. The ABFT protection of the array has the same effect as a decrease of 2 MB-seconds in the value of $\mathbf{V}_d$. So the net combined effect is to increase the program vulnerability by 1 MB-second, worsening the fault tolerance of the application.

5. Summary

The contribution of this paper is to define two metrics $\mathbf{V}_d$ and $\mathbf{V}_c$, that can be used to measure the vulnerability of the code and data spaces of an application to the effects of SEUs. These metrics are important for at least 4 reasons.

1. They form an important determinant in predicting to what extent an application will be completely unaffected by SEUs or injected faults.
2. They allow comparison between programs of their degree of vulnerability.
3. They help developers measure the effect code modifications on vulnerability.
4. They assist the evaluation of tradeoffs between code and data vulnerability.

For a broader discussion of this topic, see http://hpc.jpl.nasa.gov/PEP/pls/papers/Fault_analysis.pdf

6. Acknowledgements

This work was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

Appreciation is expressed to Daniel S. Katz, E. Robert Tisdale, and the anonymous reviewers of the paper originally submitted to the conference, for their reviews and comments.

7. References

[1] D. S. Katz and P. L. Springer, "Development of a Spaceborne Embedded Cluster," IEEE International Conference on Cluster Computing (CLUSTER2000), Chemnitz, Germany, November 2000.

[2] H. Madeira and J.G. Silva, "Experimental Evaluation of the Fail-silent Behavior in Computers Without Error Masking," *Int. Symp. Fault-Tolerant Computing, FTCS-24*, IEEE, Jun. 1994, pp. 350-359.

[3] R. Koga, W. A. Kolasinski, and M. T. Marra, "Techniques of Microprocessor Testing and SEU-Rate Prediction," *IEEE Transactions on Nuclear Science*, Vol. 32, No. 6, December 1985, pp. 4219-4224.

[4] A. Benso, M. Rebaudengo, L. Impagliazzo, P. Marmo, "Fault-List Collapsing for Fault-Injection Experiments," *1998 Proceedings Annual Reliability and Maintainability Symposium*, 1998, pp. 383-388.

[5] P. E. Lewkowicz and L. J. Richter, "Single-Event Upsets in Spacecraft Digital Systems," *ISA Transactions*, Vol. 24, No. 4, 1985, pp. 45-48.